NASA-TM-112652

NASA
7N-61-TM
021690

[01/01/1990]
No Report Date

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

ARC 275 (Rev Mar 93)

RNS-90-002

Experiences with the Fair Share Scheduler

Toby Harness

NASA Ames Research Center
Numerical Aerodynamic Simulation Systems Division
Moffett Field, California

Abstract

The intent, function, and configuration of the Fair Share scheduler is examined. System usage characteristics of the Numerical Aerodynamic Simulation (NAS) Cray-2 and Cray Y-MP and the manner in which Share was employed to address the more refractory aspects of this usage are discussed. Some modifications to the system as provided by Cray Research (CRI) were necessary to further our goals for Share. Comments are provided on some of the bugs and misfeatures encountered.

Introduction

The Fair Share Scheduler (Share) is a process scheduler that runs "on top" of UNIX's[1] scheduler in an attempt to distribute system resources more equitably. It does this by adjusting the scheduling priorities of all processes on a regular interval, based on recent resource usage and entitlement of their owner. Thus Share attempts to be fair to users, not processes or jobs. User entitlement to system resources is set by the system administrator based on a hierarchical model that permits resource allocation at each level of the hierarchy.

Share was originally developed at the University of Sydney and the University of New South Wales in the early 1980s to better support interactive use of DEC VAX 11/780s running AUSAM, a local version of UNIX [1]. Share is currently a product of Softway Pty Limited as part of their *Share* resource management system, previously known as *Limits* [2]. Since the 5.0 release of UNICOS[2], Cray Research has provided its implementation of Share [3]. In most respects it is very similar to the version described in the January 1988 issue of the *Communications of the ACM* [1].

[1]UNIX is a registered trademark of AT&T.
[2]UNICOS is a registered trademark of Cray Research, Inc.

How Share Works

Resource Groups

Resource groups are used to allocate system resources, and are unrelated to traditional UNIX groups. An allocation of system resources is referred to as a *share*. All users must be members of exactly one resource group.

A resource group may be a member of a parent resource group, and so on in turn, up to the system-wide resource group, known as *Root* (note the capital letter). As one might expect, this resource group has the entire system allocated to it. This forms a Share tree, with *Root* as the base of the tree, other resource groups as intermediate nodes, and users as leaves. See Figure 2 for an example. Every process in the system is *attached* to a node of this tree. These nodes are referred to by the name of the kernel data structure used to represent them, limit nodes, or *lnodes*.

Shares

Both resource groups and users are allocated shares from within their parent resource group. The number of shares they are allocated in relation to the number of shares other members within their resource group are allocated determines the percentage of their parent's share to which they are entitled. Note that between two members of the same resource group it is the ratio of shares that is significant, not the numbers of shares. For example, within a given resource group, allocating a privileged user 500 shares when every other user is allocated 100 is the same as giving this user 50 shares if every other user has only 10.

Because shares are allocated at the resource group level, shares cannot be directly compared between members of different resource groups. A comparison can only be done with normalized shares.

Shares are never consumed, but rather remain constant until the system administrator changes them.

Usage

Since shares are not consumed, there needs to be some measure of how much of the system a given user or resource group has used in order to make sure everyone is getting their fair share. This measure is called *usage*. It is a unitless number that is calculated from the amount of CPU time used, memory used, blocks of I/O read or written, and number of system calls made by all processes attached to a given lnode.

Resource use is exponentially decayed over time, with a configurable half-life. The length of the usage half-life has the greatest impact on the users of the system. Long half-lives – more than a few hours – favor constant resource consumption, typical of batch processing, and short half-lives – less than an hour or so – favor "burst" consumption that is more typical of interactive processing. See Figure 1.

Usage, like shares, is a relative term: its absolute magnitude is not as significant as its normalized value in comparison to the usage accumulated by other users and resource groups. There is a configurable maximum usage parameter that prevents very large usages from distorting Share calculations.

Priorities

Share attempts to minimize the disparity between usage and share entitlement by reducing the priorities (i.e., increasing the numerical value) of processes associated with users with higher normalized usage than normalized share. This is Share's only interaction with the low level process scheduler.

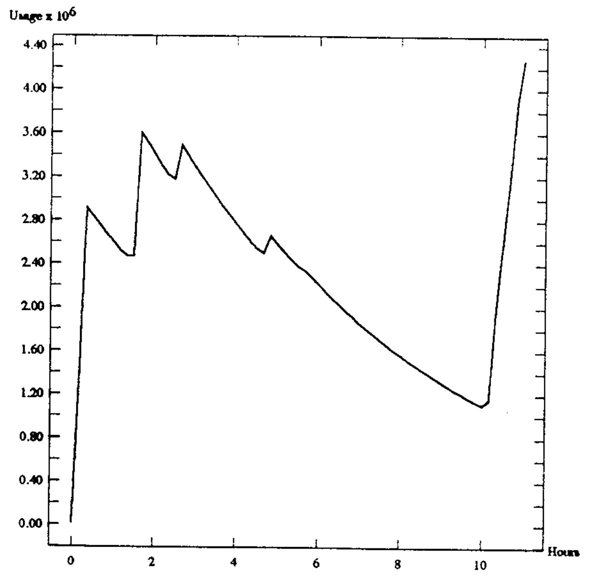

Figure 1: *An example of usage decay. The half-life is two hours.*

Competition

Share requires competition to be fair. A user or resource group that is entitled to a percentage of the system may not receive its full entitlement if it does not generate sufficient demand. Since unused system resources are never wasted, a user or resource group is allowed to use more than its share of the system if other users or resource groups are not using all that they are entitled to.

The idle processes are attached to the *Idle* resource group, which has zero shares allocated to it, and only run when there is insufficient global demand.

Readers are directed to [1] for a more detailed discussion of the workings of Share.

System Usage Characteristics

In an attempt to circumvent the Network Queueing System (NQS) in order to get quicker turnaround, NAS users were simulating batch processing by chaining together interactive runs. These "interactive batch" jobs consumed large amounts of CPU time and overcommitted main memory, causing excessive daytime swapping. Only a few recalcitrant users were responsible for the bulk of this type of processing. The CPU cycles consumed by these jobs, plus the overhead of the additional swapping, further slowed NQS turnaround. This negative feedback loop not only overcommitted system resources during prime time, but also frequently resulted in system underutilization during off hours due to the low number of jobs placed in the NQS queues.

This undesirable situation was possible in part because of the liberal interactive limits we enforce. Current limits are 600 seconds of CPU time and 20 Megawords of memory per process on the Cray 2/4256, and 600 seconds and 10 Megawords on the Cray Y-MP/8128. These limits are necessary to support the type of intrinsically interactive applications we consider important, such as *plot3d*, and we sought alternatives to lowering them.

Share Configuration

Share was selected as a mechanism to distribute system resources more equitably among interactive users and to provide a guaranteed percentage of CPU cycles to NQS jobs. The NQS scheduler could then manage a more predictable system. To accomplish this, Share was configured to support different resource allocations for interactive and batch processes, and suitable modifications to NQS (8 additional lines of code) were made so that NQS could properly interact with Share.

Hierarchy

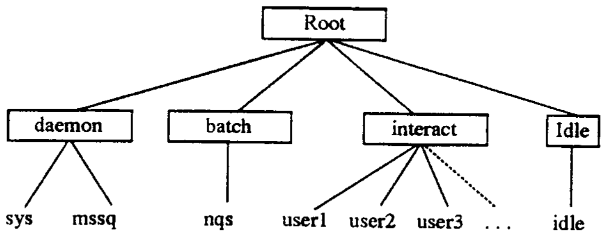

Figure 2: *Share Hierarchy*

We initially created two resource groups, *batch* and *interact*, directly under *Root*. We later added the *daemon* resource group to prevent system process from taking 100% of the machine in the event of serious problems. See Figure 2.

All system processes, except for the Share daemon itself and the idle processes, run under *daemon*, which is allocated a 90% percent share. This is to ensure these processes receive the resources they need. All interactive user processes run under *interact*, with an equal share. All batch processes – those started by NQS – run under the *batch* resource group as if initiated by a pseudo-user created for this purpose called *nqs*.

The *nqs* pseudo-user is used only for Share calculations. In particular, the user, group, and account identifiers of the batch processes are unchanged. This is done by attaching NQS jobs to the *nqs* lnode.

This means that users do not incur any of the share charges accumulated by their batch jobs. *User's interactive sessions are thus unaffected by the resource consumption of their batch jobs.*

Because all NQS jobs (actually, their constituent processes) are connected to the same lnode, Share does not differentiate between NQS jobs. Thus Share's role is largely relegated to mediating between NQS processes *en masse* and individual interactive processes. (The *daemon* and *Idle* resource groups do not normally consume enough system resources to warrant consideration.)

Share could be made to differentiate between NQS jobs by having NQS attach each job to a unique "transient" lnode under *batch* instead of to *nqs.*[3] These transient lnodes could have shares assigned to them based on either the priority of their originating NQS queue or the entitlement of their submitter, or some combination. We have not needed to do this, as we assign equal shares to all users and NQS queue *nice* values are sufficient for propagating queue priorities.

Numbers

Despite the ability to charge for various types of resources, we have elected to charge only for CPU ticks at the arbitrary rate of 100 (this is a unitless number) per tick.

The only other resource we gave any serious consideration to charging for was memory utilization. Ultimately we decided against this until we had more experience with Share and could better predict its impact. Since then we have become leery of having

[3]Suggested by Andrew Bettison on the ShareSIG mailing list.

large memory jobs with low priorities in the system. Note however that this charge accumulates very rapidly, and any non-zero rate needs to be 3 or 4 orders of magnitude less than the CPU rate if it is not to dominate the resource usage calculations.

We chose a large maximum usage of 1×10^9 so as not to negatively impact *nqs*, which tends to run up a large usage. Processes attached to lnodes with usages close to the maximum usage are disfavored by Share.

The usage half-life is 2 hours, which encourages load spreading throughout the day. Figure 1 is an example of this half-life's effect on the usage of an early morning interactive user.

We also keep a tight reign on the maximum effective share an individual user can have (1.25 times allocated share) and the minimum effective share a resource group can receive (0.9 times allocated share).

All these configuration parameters are setable via **shradmin**; the entire configuration line as it appears in our `/etc/rc` is `/etc/shradmin -F06 -R4 -K2h -t100 -U1e11 -X1.25 -Y.9`

Shares

The only other significant numbers in our configuration are the number of shares we allocate to *batch* and *interact*. During prime time on the Y-MP these are 70 and 30, respectively; they are 60 and 40, respectively, on the Cray-2. On both systems these are changed to 90 and 10 for non-prime time. Prime time for us runs from 0400 to 1730, as we have users on both coasts.

When shares are dynamically changed, it is important to zero out usage for the effected lnodes (the *batch* and *interact* resource groups in this case), otherwise it may take some time for their usages to adjust to the newly desired values. By discarding past usages that were accumulated under a different set of rules, all resource groups and users start out evenly.

The magnitude of shares to allocate resource groups and users should be large enough to permit a reasonable granularity. Otherwise it may not be possible to adjust a share allocation to a desired value without adjusting a potentially large number of allocations. We find picking values that sum to 100 to be convenient.

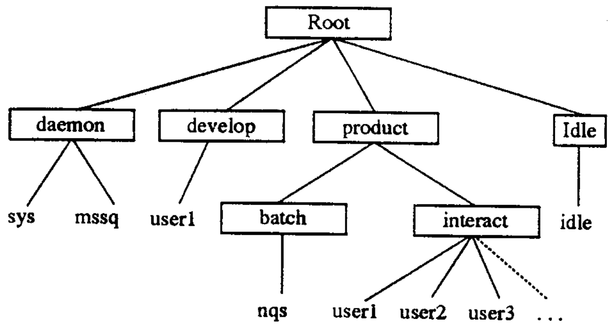

Figure 3: *Modified Share Hierarchy*

One benefit of the flexibility Share provides was demonstrated during the acceptance period for a memory upgrade to our Cray Y-MP. When high priority acces to machine resources were needed for running acceptance tests, *batch* and *interact* were put under a production resource group, *product*, and another new resource group, *develop*, was created. See Figure 3. This resource group was given a large share of the system, which allowed the acceptance tests to run virtually unconstrained while permitting production use to continue.

Code Modifications

The modifications necessary to NQS to support this type of application of Share are trivial. All that is necessary is to attach the newly created or restarted job to the *nqs* lnode instead of the owner's lnode. This is done with the **setshares** library call in **nqs_spawn.c** for restarted checkpointed jobs and in **nqs_reqser.c** for new jobs. As attaching a process to an lnode is a privileged operation, this must occur prior to changing the uid of the new process from super user to the owner's.

A mechanism for changing the shares and usage for a resource group or user without rebooting the system was not provided by CRI. We wrote a general purpose tool that uses the new **limits** system call to dynamically alter the fields of an active lnode. We then added a **cron** entry to invoke this at the appropriate times to switch between prime and non-prime periods.

Finally, we altered **su** to attach a successfully created root process to the *Root* lnode, instead of leaving it attached to the invoker's lnode.

Bugs and Misfeatures

We consider the supplied behavior of `su` to be incorrect. A critical system process that is started or restarted inadvertently attached to a regular user's lnode may not get as much system resources as it needs, and what it does get will be added to the invoker's resource use.

CRI does provide the `shrlimit` command for attaching a new process to a different lnode. However, this had a simple bug (actually in `setshares`) that kept it from attaching to `root`'s lnode (which is *Root*.) Although this was easily fixed, we thought it safer to modify `su` than to rely on our privileged users to always remember to use `shrlimit`.

Although the system "autoconfigures" *Root*, *Idle* is not similarly set up. If *Idle* is not properly configured with zero shares, the idle processes will be inappropriately scheduled to run.

Resource groups appear in **/etc/passwd**. Although not users, resource groups have a **udb** entry, and **udbgen** dutifully includes them in **/etc/passwd** and **/etc/group**.

The default number of shares **udbgen** assigns to a user is zero. This is not a particularly useful allocation.

A user may belong to only one resource group. It would be useful to be able to enroll a user in more than one resource group, and provide a means to switch between them. Only one resource group would be active at a time. This would work much like the old **newgrp** system.

Perhaps the biggest problem with Share as distributed by CRI is the documentation. The manual pages in particular are written in a confusing style that uses inconsistent – and sometimes incorrect – terminology. The System Administration Guides are better, but still fail to define some important terms. Documentation for the `libshare` library routines is completely missing.

User Acceptance

When we enabled Share in the above configuration in December of 1989, there was some confusion and consideration among our users about how the new scheduler would impact them. Although most users were unfamiliar with the functional parameters and behavior of the low level process scheduler that they had been using for years, many felt they needed to understand the workings of Share.

One year after turning it on, most users are aware that Share is running, but remain unfamiliar with how it works. Interactive graphics users who had all but given up on daytime use of the systems are pleased with their improved response. NQS throughput is still not as high as most users would like, but outside of periods of low demand for the interactive share of the system, "interactive batch" users have largely switched back to using NQS for long running background jobs.

Conclusion

In general, the intergration of Share into UNICOS appears not to have been well examined by CRI. However, our experiences with the Fair Share scheduler, once suitably configured, have been very satisfactory. Swapping was reduced dramatically, and NQS throughput improved and became more predictable. In addition to solving the immediate problems in an elegant and egalitarian manner, Share provided us with new functionality and flexibility.

References

[1] Kay, J., and Lauder, P. A Fair Share Scheduler. *Communications of the ACM* vol. 31, 1 (Jan. 1988), 44–55.

[2] Bettison, Andrew, et al. Limits — A System for UNIX Resource Administration. *Proceedings of Supercomputing '89* (Nov. 1989), 686–692.

[3] Cray Research, Inc. UNICOS System Administrator's Guide for CRAY-2 Computer Systems (SG-2019 C-01) 6-12 - 6-25.